

AN ITERATED SUBDIVISION ALGORITHM FOR PROCEDURAL ROAD PLAN GENERATION

Nicholas Rudzicz

School of Computer Science, McGill University

Nicholas.Rudzicz@mail.mcgill.ca

Clark Verbrugge

School of Computer Science, McGill University

clump@cs.mcgill.ca

ABSTRACT

The generation of detailed virtual environments is an increasingly resource-consuming task for videogame developers. This has encouraged the investigation of procedural techniques for creating content from landscapes to textures to—much more recently—cities and their constituent road-scapes. This paper introduces the *Iterated Subdivision* algorithm, a straightforward, flexible, and easily customised approach to the generation of road plans for virtual cities. The use of such an algorithm results in a significant savings in terms of developer time and resources—extending the possible scope of games—and furthermore allows rapid prototyping in order to test newly-created game assets.

INTRODUCTION

As both the graphical power and storage capacity of modern computers become steadily cheaper and more powerful, video games are driven towards greater levels of scale and detail. However, this comes with an associated increase in game development costs, both in terms of time and developer resources. In order to meet expectations, game developers are required to hire additional artists, buy additional toolsets, and spend larger fractions of a game’s budget simply to provide the game world with sufficient environmental content. Accordingly, there has been a growing interest in procedural generation of a variety of in-game assets, including whole cityscapes. The latter present an interesting problem, as cities and human settlements in general tend to exhibit much more regular and meaningful patterns than can be found in natural environments. As such, any attempt to generate them procedurally (i.e., randomly) must nevertheless maintain the appearance of some underlying structure.

The procedural generation of cities can itself be decomposed into several smaller tasks: road plan generation, generation of buildings and green spaces, population with NPCs and vehicles, and so on. The first of these items—and indeed, generally considered the most important and initial step in city generation—is the construction of a road plan upon which the city can

be built. Here we present an *Iterated Subdivision* algorithm designed for this task. Our approach is fast, simple to conceptualise, and can easily be tuned according to the specific requirements of the game environment. By including simple constraints and exploiting our tuning parametrisations we can produce a variety of road plan styles suitable for city simulation. This design produces realistic results, and is particularly suited to the rapid development of large-scale game worlds as well as to rapid prototyping for game testing purposes, or even on-line road plan generation.

Specific contributions of this paper include:

- We develop a simple algorithm for road plan generation suitable for game development. This approach is extremely fast and accommodates arbitrary city boundaries.
- To further improve realism we incorporate various constraints that enhance output appearance. Our generated road plans are capable of modelling both structured and unstructured city designs.
- Different game environments mean a wide variety of road properties may need to be supported, and moreover these may not be constant throughout the city. Our technique allows the use of density map information to represent arbitrary, spatially-localised parametrisation.

The next section describes related work in the field of content generation in general, and city generation in particular. This is followed by the main section of this paper presenting our Iterated Subdivision algorithm, along with quality constraints and a number of parametrisation options, and showing representative output. In the final section we discuss possibilities for improvement to the algorithm design as well as future work on the software itself.

RELATED WORK

Although procedural content generation is the subject of an increasing amount of research in recent years, it is not entirely new to the field of videogame development. Early “dungeon-crawlers” such as *Nethack* produced randomly-generated dungeons for the player to

navigate, and both Adams and Buck have performed similar work in recent years [Adams, 2002, Buck, 2003]. These approaches each consider game dungeons as sets of “interesting” rooms connected by either a maze or graph topology, and set about examining the various (random) ways that these connections can be made.

Similarly, procedural landscape generation has been popular for many years and has achieved a certain level of sophistication. The popular and enduring *Civilization* series of games provide randomised terrain and settlements based on a small set of input parameters selected by the user. Though these terrains generally belie the grid-based engine on which they are built, other techniques—as outlined by Ebert—employ algorithms such as Perlin noise, fractals, or displacement and erosion to generate continuous, highly realistic terrains for visualisation purposes [Ebert et al., 2002].

However, while games like *Nethack* and *Civilization* have been able to significantly increase their longevity through the use of randomised content—ensuring a new gaming experience each time they are played—commercial videogames employing procedural content (whether implicitly or explicitly) are still vastly outnumbered by those featuring traditional, manually-created content. In the case of procedural city generation, the greatest share of work is instead to be found in the fields of visualisation and simulation. Here, various levels of content are often generated: streets, buildings, green spaces, and so on; however, each approach requires an initial road plan to be generated in order to give the remaining steps a structure on which to build. Generally, algorithms for this road plan generation have taken the form of L-Systems or agent-based approaches.

L-Systems

One of the earliest and perhaps most successful approaches to procedural road plan generation involves the use of Lindenmeyer Systems, or *L-Systems*. In its simplest form, an L-System is a deterministic string rewriting algorithm: given an alphabet of symbols, a set of production rules, and a starting string (axiom), the algorithm uses the rules to repeatedly and simultaneously replace multiple substrings with new strings. Using this process, it is possible to realistically model branching structures, as shown by both Lindenmayer [Lindenmayer, 1968] and Prusinkiewicz [Prusinkiewicz et al., 1988] in their modelling of cell and plant growth, respectively.

Parish and Müller build on the work of Lindenmayer and Prusinkiewicz, using L-Systems to model the growth of urban road networks [Müller, 2001, Parish and Müller, 2001]. Their L-System implementation is extended, however: first, the L-System is

“self-aware” in that it can form closed loops with itself (i.e., new roads can create cycles by connecting with pre-existing ones); and second, the L-System is highly parametrised—for instance, it can be made to follow population density or terrain elevation, or follow more grid-like patterns as opposed to radial ones. This modified L-System approach has proven very successful in generating realistic road networks [Parish and Müller, 2001].

Agent-based approaches

Lechner and Watson explore the use of autonomous agents in the creation of road networks [Lechner et al., 2003, Watson, 2006]. A collective of independent “developer agents” is built, with each individual being assigned a specific type of urban content—roads, residential areas, industrial plots, and so on. Each type of agent is attracted and repulsed by different characteristics of the underlying terrain and existing city objects (residential agents are drawn to waterfronts, for instance, while road agents are drawn to unconnected lots), and when the agents are “released” into the world, cities and road plans emerge from their interaction and competition for virtual real estate and resources. While this model perhaps most accurately represents the dynamic growth of cities under competing forces, the authors admit that the results are presently very coarse-grained and on a much smaller scale than desired [Lechner et al., 2003]. Furthermore, as in any agent-based approach, results are highly unpredictable, though Watson has suggested means for users to significantly influence the final product by placing strong attractors (or “honey”) in various areas of the virtual world [Watson, 2006].

ITERATED SUBDIVISION

While the previous methods for road plan generation have proven successful in a number of ways, they come at the cost of programming complexity. Our approach to the problem is based on an efficient and conceptually simple method that nevertheless retains the flexibility and realism exhibited in previous algorithms. Since this approach relies exclusively on the repeated subdivision of polygons, it has been dubbed the *Iterated Subdivision* (ItSub) approach. This initial algorithm is extended first through the imposition of various internal constraints to ensure realistic outputs, and finally by allowing custom parametrisation of the output through a modular use of bitmaps. The following subsections describe these steps in detail.

Algorithm 1: Iterated Subdivision

Input: Polygon P , A_{min} , A_{max}
Output: S_{allot} , S_{roads}

```
1  $S_{oversized} \leftarrow P$ 
2  $S_{roads} \leftarrow \emptyset$ 
3  $S_{allot} \leftarrow \emptyset$ 
4 repeat
5    $P_{working} \leftarrow S_{oversized}.pop()$ 
6    $L_{bisect} \leftarrow P_{working}.getAcceptableBisector()$ 
7    $\{p_1, p_2\} \leftarrow P_{working}.bisect(L_{bisect})$ 
8    $S_{roads} \leftarrow S_{roads} \cup L_{bisect}$ 
9   if  $p_1.area > A_{max}$  then
10    |  $S_{oversized} \leftarrow S_{oversized} \cup p_1$ 
11  else
12    |  $S_{allot} \leftarrow S_{allot} \cup p_1$ 
13  end
14  if  $p_2.area > A_{max}$  then
15    |  $S_{oversized} \leftarrow S_{oversized} \cup p_2$ 
16  else
17    |  $S_{allot} \leftarrow S_{allot} \cup p_2$ 
18  end
19 until  $|S_{oversized}| == 0$ 
```

Algorithm

The Iterated Subdivision algorithm draws inspiration from Tarbell’s *Substrate* visualisation, which was noted to produce “intricate city-like structures” [Tarbell, 2008]. This result is achieved by placing a number of random seed vectors on a two-dimensional “canvas,” and repeatedly drawing roughly perpendicular line segments from these initial vectors and other newly-created segments. Appearance and quality are driven by the method for assigning new line-drawing vectors, and the allowable range of sizes for the resulting space division.

In its simplest form, the *ItSub* algorithm requires as input a predefined, simple polygon P , and a minimum and maximum area— A_{min} and A_{max} , respectively—for the resulting subdivisions, or allotments. We discuss further parametrisation below. The algorithm then bisects P randomly, resulting in two new polygons. The latter are either accepted or rejected based on their individual areas: larger polygons are subdivided further, while those of an acceptable size are set aside as completely subdivided.

The process just described is summarised in Algorithm 1. The key component of the algorithm is the set of “oversized” polygons, $S_{oversized}$, containing all polygons with an area greater than A_{max} . Initialised with only the original polygon P , $S_{oversized}$ returns polygons one at a time. A random bisecting line, L_{bisect} , is generated within this working polygon by choosing a random edge

in the latter, and a random point along this edge. A random line drawn through this edge (not parallel to the edge itself) will then bisect the polygon and return two new polygons, p_1 and p_2 . These resulting polygons are evaluated individually. First, if either polygon is too small (having an area less than A_{min}), the bisecting line is rejected and a new one generated; otherwise, L_{bisect} is accepted. Then, if a polygon is still oversized, it is added back into $S_{oversized}$, while if its size is acceptable, it is added to S_{allot} , which represents the final set of all generated polygons, or allotments. Likewise, L_{bisect} is added to S_{roads} , the set of all road segments generated by the algorithm; note that, while S_{roads} could theoretically be rebuilt from the knowledge contained in S_{allot} and vice-versa, the two are stored explicitly to facilitate any subsequent post-processing steps. The algorithm then proceeds until there are no more oversized polygons in $S_{oversized}$ to process.

Acceptable polygons

The choice of bisecting lines is clearly critical to the visual success of the algorithm, and various constraints can be applied to the *ItSub* algorithm in order to ensure realistic road plan generation. Minimally, A_{max} is necessary to ensure that the algorithm eventually terminates, and A_{min} is used primarily for aesthetic reasons, to ensure allotments, the spaces between roads, are above some minimum size. A further “shape” criterion is also necessary; even if they cover sufficient area allotments should typically have a usable shape, in accordance with realistic city design.

The result of a naive implementation of *ItSub*, without any shape constraint, is shown in Figure 1. Here, the generated polygons satisfy the condition of being smaller than the supplied A_{max} ; on the other hand, it is clear that the algorithm is generating too many long, overly narrow “stripes.”

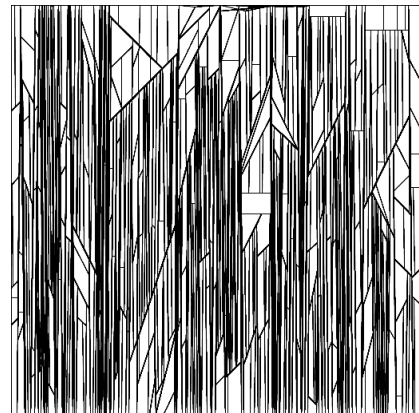


Figure 1: Unconstrained *ItSub* output displaying undesirable “stripes”

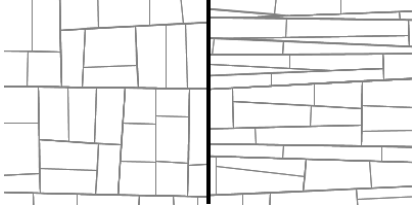


Figure 2: Comparison of allotments with small diameter-to-width ratios (left) and larger ones (right).

Imposing a shape constraint requires reducing the range of “acceptable” polygons returned by the `Pworking.getAcceptableBisector()` function. Previously, it was sufficient that a polygon P_i have an area larger than A_{min} . To remove the appearance of stripes, however, it is necessary to examine the ratio between the largest and the smallest spans of the polygon—that is, the ratio between the polygon’s diameter and width, as defined by both Preparata and Toussaint [Preparata and Shamos, 1985, Toussaint, 1983]. As this ratio approaches infinity the polygon becomes increasingly “striped” and elongated; conversely, as it approaches unity, the polygon is contracted. Acceptable polygons are thus redefined to be those that are larger than A_{min} , as well as having a bounded diameter-to-width ratio below some threshold R_{max} . The effects of varying R_{max} are demonstrated in Figure 2; tests have shown that using $R_{max} = 16$ provides acceptable results, and this value is used in all subsequent figures.

Branching angles

The previous description of the `ItSub` algorithm suggested that a bisecting line (providing it is “acceptable”) is chosen to pass through a polygon at a random point, and at a random angle. In practice, however, this leads to completely arbitrary road paths, as seen in certain areas of Figure 1, particularly near the top of the image. While a certain amount of arbitrariness is expected in urban road plans, it is unrealistic when evident in any large proportions. To avoid this, branching angles can be constrained to lie within specific intervals. In the current implementation, branching angles are initially assumed to be perpendicular to the starting edge, and are then perturbed by a small amount in either direction. The perturbation is defined by a Gaussian distribution as shown in Figure 3, whose parameters—mean and standard deviation—can be defined at runtime. As discussed elsewhere (see [Parish and Müller, 2001]), many modern cities exhibit more than one distinct “style” of road plan—large-scale patterns emerging from the specific orientation of a collection of adjacent roads. The manipulation of branching angles described above allows for at least two styles of road plan to be generated with `ItSub`: *Manhattan* and *Arbitrary*, as shown in Figure 4.

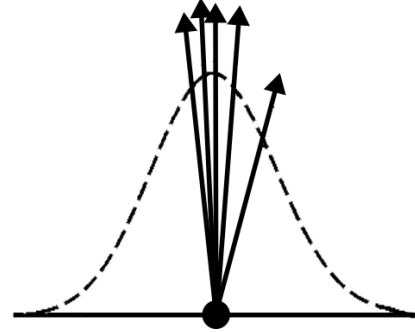


Figure 3: Gaussian distribution of possible branching angles

Manhattan-styled roads, named after the city in which they are most conspicuous, exhibit a grid-like pattern, with angles that rarely deviate from 90° . Such a road plan can be achieved by setting the standard deviation, σ , of the aforementioned Gaussian distribution to zero. Thus, effectively all roads will branch at right angles to their starting point, giving the grid-like effect intended. These types of roads are generally prevalent in modern, rigorously-planned or commercial neighbourhoods.

Conversely, arbitrary road plans are more typical of older neighbourhoods, in which urban planning was of much less importance than simple expedience; examples can be seen at the southern tip of Manhattan island, as well as in Montreal’s Old Port district. Producing such roads using `ItSub` is again achieved by manipulating the value of σ as described above. For Arbitrary road patterns, σ is significantly increased, which leads to a wide variety of road branching angles, as required.

USER INPUT

The discussion of `ItSub` thus far has ignored the issue of user input; all parameters have been assumed to be fixed at runtime. It is, however, advantageous—perhaps even necessary—to allow these parameters to vary within a particular generation run in order to produce an urban road plan with a variety of features. Given the visual

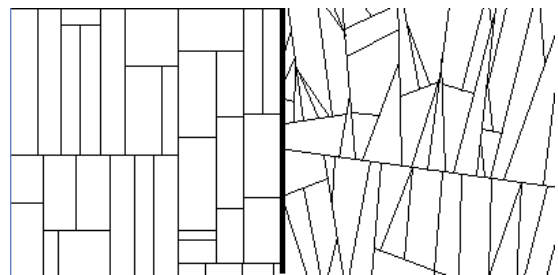


Figure 4: Two styles of road patterns: Manhattan (left) and Arbitrary (right)

nature of the generation task, and similar to other techniques in the literature, our approach makes use of input bitmaps to provide this flexibility to the user. These bitmaps are standard greyscale images, as shown in Figure 5, that can be custom-drawn by the user, and whose purpose is to illustrate the variation of a specific trait—such as population density or characteristic road pattern—across a city map.

Population density

The vast majority of cities display some degree of variation in population and road density patterns across the urban area; in fact, it would be challenging to find a city that does not exhibit such variation. Previously, however, our *ItSub* algorithm has assumed uniform density during the generation process. In order to overcome this constraint and allow users to define regions of higher or lower density within a given city, greyscale bitmaps are used as input—lighter areas indicating high density, darker areas indicating low density.

Note that adding steps to the *ItSub* algorithm to account for user-defined density maps does not significantly alter the technique; the trivial case, where no density map is provided, is identical to the algorithm described above. If the input is given, however, some extra processing is required during the polygon-testing phase. Whereas previously, a newly-generated polygon was compared against A_{min} and A_{max} directly, these values must be somewhat modified upon the introduction of a density map. First, an axis-aligned bounding box is created around the new polygon. The boundaries of this box (defined by the vertices of the polygon) are then mapped onto the bitmap, and the average of the greyscale values within this box is calculated. Finally, this average is used as an inverse weight on A_{min} and A_{max} —polygons mapped onto a lighter area of the density map will be compared against *smaller* values of A_{min} and A_{max} , resulting in much shorter and densely-packed roads in these areas. Conversely, polygons mapped to darker areas will be compared against

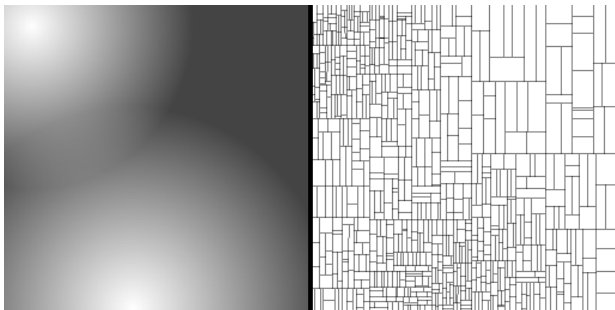


Figure 5: The use of a greyscale density map (left) influences the size of generated allotments (right).

larger bounding values, and will tend to be subdivided less, resulting in larger allotments. The use of a density map is shown in Figure 5.

Road pattern

The choice of road pattern can also be influenced by user-defined bitmaps in a similar fashion. As before, a given greyscale bitmap is used to describe the influence of a given road pattern (Manhattan or Arbitrary) over a particular area, ranging from low influence (dark) to strong influence (light). Note that in the case of road pattern bitmaps, the parameter to be controlled is the branching angle of a road from a specific point, as opposed to the overall area of a full polygon, as was the case with density maps. As such, instead of mapping the bounding box of the working polygon onto a bitmap, it is rather the branching point itself that undergoes mapping. A small selection of pixels in the surrounding area in the bitmap are examined, and an average greyscale value is computed. This value then determines the influence of a particular road pattern on the given branching point.

Importantly, there are now two such bitmaps to evaluate. The procedure described above is carried out on both the Manhattan- and Arbitrary-influence bitmaps, and the resulting averages compared. The new branching angle is then chosen to reflect the pattern that has the greatest influence on the branching point. The use of a road pattern bitmap is demonstrated in Figure 6.

ANALYSIS

Given the full definition of the *ItSub* algorithm, it is beneficial to examine the procedure’s performance. To do this, experiments were conducted in which the total number of road segments generated was gradually increased, with the expectation that computation time will be roughly linear in the number of roads generated—since exactly one segment is produced per

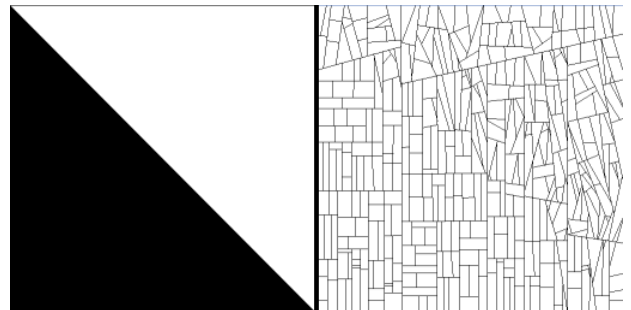


Figure 6: A bitmap showing the distribution of Arbitrary-styled roads (left) and the resulting road plan (right).

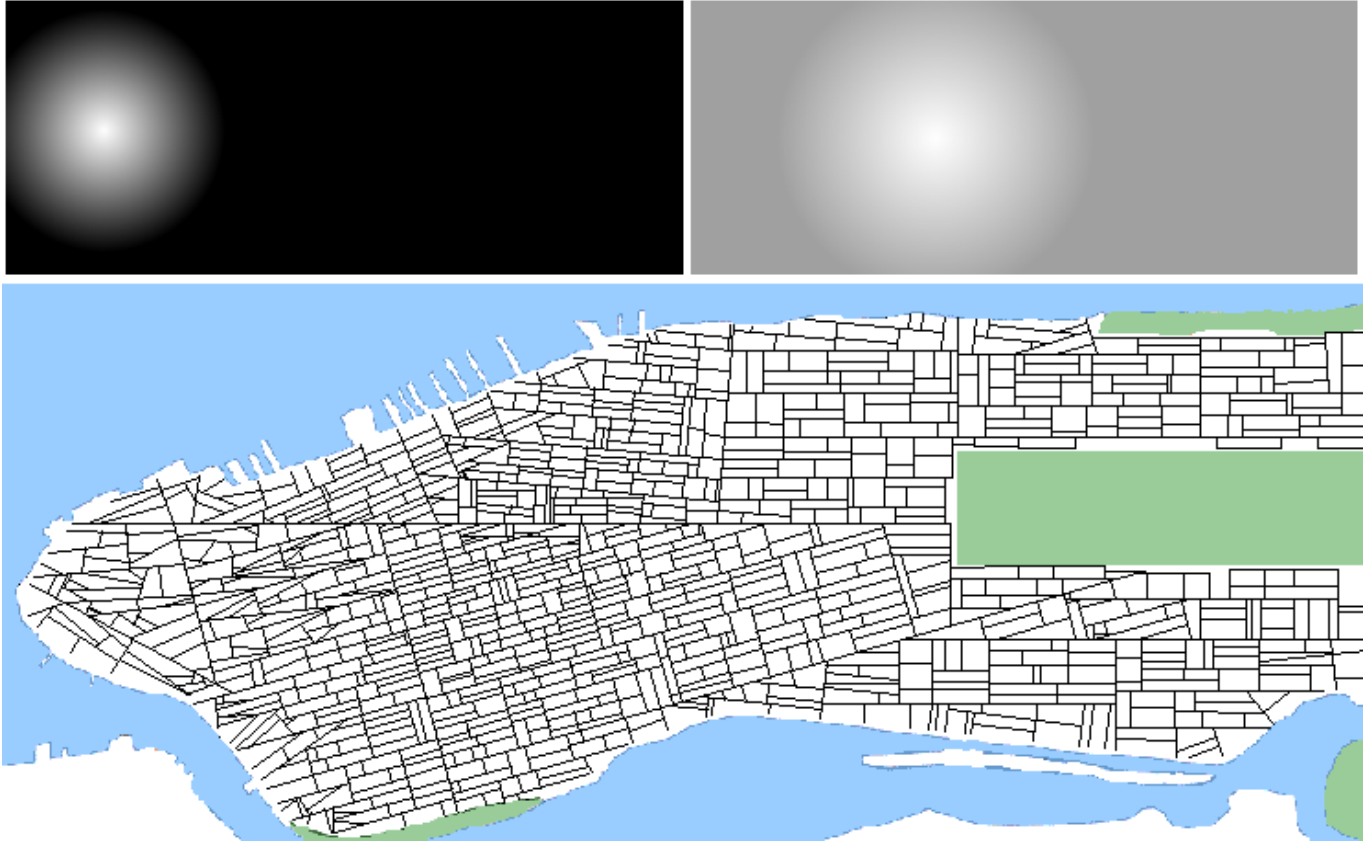


Figure 7: Results of applying a simple implementation of ItSub to a representation of Manhattan island. Top-left: bitmap representing Arbitrary road plan distribution—see arbitrary branching angles towards left side of the image. Top-right: bitmap representing density distribution—light grey background ensures a minimal density in all areas. Bottom: resulting road plan. Greenspace is constructed by removing generated road segments within a given area.

polygon examined, and there are no comparisons to perform between polygons, etc. Operating on initial polygons of varying size (512x512 pixels, and multiples of 5, 10, and 20 times this polygon), road plan generation was executed a number of times, and the overall number of road segments generated were averaged, along with total computation time. This experiment was conducted first on the base algorithm using none of the three input bitmaps (density, Manhattan, or Arbitrary), and then with all three included, to determine whether the calculations involved in examining these bitmaps would significantly affect running time. Experiments were conducted on an AMD Sempron 1.6Ghz notebook with 768MB of RAM running Xubuntu 7.10. Results are shown in Table 1.

As observed in this table and the associated graph, running time increases roughly linearly as more roads are generated. Furthermore, as expected, the use of input bitmaps initially increases running time of the algorithm. Interestingly, in the case of the two larger maps running time was either unaffected by bitmap usage or actually decreased by nearly a full second. This can be explained by examining the number of roads gener-

No bitmaps		All bitmaps	
Roads	Time (ms)	Roads	Time (ms)
200	386	60	944
1015	1190	301	1710
2043	2437	639	2459
4074	4392	1366	3521
—	—	4259	9877

Table 1: Influence of number of roads generated on algorithm running time.

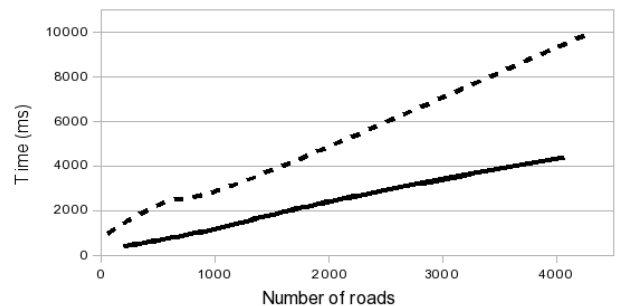


Figure 8: Number of road segments vs. algorithm run time (from Table 1). Solid line shows tests using no bitmaps, dashed line shows tests using all bitmaps.

ated: when a density map is used, fewer road segments are generated due to the existence of sparser areas, and so the overall running time of the algorithm is shorter. Thus, while the use of input bitmaps does incur some overhead, this can be effectively negated when using density maps. Note that the number of actual roads generated is not directly under user control, and is instead a consequence of bitmap constraints as well as the size of the initial polygon. Nevertheless, the algorithm maintains an exceptional speed: the road network pictured in Figure 7 was regenerated multiple times, averaging 1200 individual roads and running for 2.25 seconds on average.

CONCLUSIONS & FUTURE WORK

The combination of simple constraints and bitmap parametrisation makes the Iterated Subdivision algorithm very modular, simple to use and modify, and scalable up to large city sizes. Basic constraints ensure basic output quality, while the bitmaps make local specialisation trivial, with variations in road plan seamlessly integrated throughout the final output. Realistic examples are shown in Figures 7 and 9, both demonstrating the results of applying the `ItSub` algorithm to real-world geography—the islands of Manhattan and Montreal, respectively.

Of course there are many areas to improve and investigate as future work. Currently, the parametrised definition of road density allows users to influence the size of generated allotments, approximating transitions between “downtown” cores and less-developed areas. However, future parametrisation might involve more explicit definitions of urban versus suburban allotments—again through the use of input bitmaps. Such input would not only influence the density of roads and allotments, but could be extended to influence the types of buildings generated in each allotment, should such a process be implemented.

One consistent difficulty in generating road plans such as those presented in this paper is the generation of an initial bounding polygon. Since `ItSub` requires an initial polygon P as input, the latter must be defined manually beforehand. Though a simple square polygon would suffice for prototyping purposes, more complex figures, such as the maps shown in Figures 7 and 9, generally require much more complex polygons to be defined. While this can be performed manually, the main purpose of developing `ItSub` is to automate the process of creating cityscapes. Thus, some work remains to be done in automatically generating these initial bounding polygons, and would likely take some inspiration from the field of pattern recognition and feature extraction.

An interesting observation of the `ItSub` algorithm is

that it lends itself extremely well to recursion and/or parallel processing: once a polygon is subdivided, each resulting sub-polygon could then be further subdivided within a separate process, or as a recursive call. It is hypothesised that a multi-threaded approach (with some considerations for a minor synchronisation issue) would provide the algorithm with a significant performance boost, while a recursive implementation would provide only negligible improvements, due to inefficiencies in maintaining a potentially large call stack. However, tests are required to explore both possibilities.

Finally, efforts are currently underway to integrate the `ItSub` algorithm into a content generation tool, in order to provide a quicker means of testing the various parametrisations described in this paper, and to observe its applicability within an actual content creation pipeline. It is being developed alongside McGill’s “Mammoth” MMOG project [MAMMOTH Team, 2008], and will soon be available for testing.

Acknowledgements

This work was supported by the Natural Science and Engineering Research Council of Canada.

REFERENCES

- [Adams, 2002] Adams, D. (2002). Automatic generation of dungeons for computer games. Undergraduate dissertation, University of Sheffield.
- [Buck, 2003] Buck, J. (2003). Random dungeon design: The secret workings of Jamis Buck’s dungeon generator. Website. <http://www.aarg.net/~minam/dungeon.design.html>, last visited Jul. 18, 2008.
- [Ebert et al., 2002] Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. (2002). *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [Lechner et al., 2003] Lechner, T., Watson, B., Wilensky, U., and Felsen, M. (2003). Procedural city modeling. In *1st Midwestern Graphics Conference*, St. Louis, MO.
- [Lindenmayer, 1968] Lindenmayer, A. (1968). Mathematical models for cellular interaction in development – i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18:280–289.
- [Müller, 2001] Müller, P. (2001). *Design und Implementation einer Preprocessing Pipeline zur Visualisierung prozedural erzeugter Stadtmodelle*. Master’s thesis, ETH Zürich.

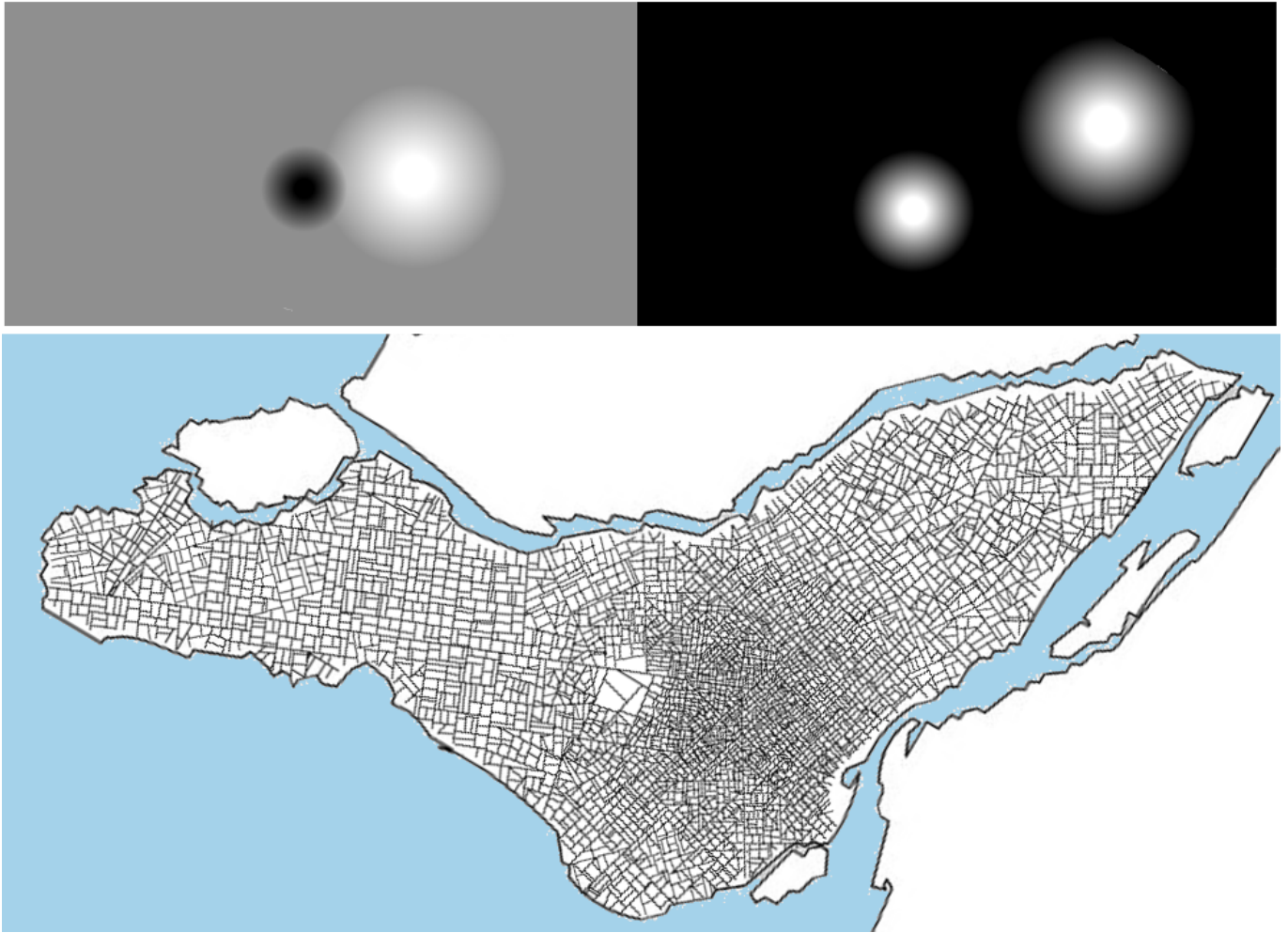


Figure 9: Application of ItSub to the island of Montreal. Top-left: Map of road density—note the correspondingly variable density in the final image. Top-right: Map of “arbitrary” road patterns—note the more random branching angles on the right half of the island, and around the area of minimal road density.

[Parish and Müller, 2001] Parish, Y. I. H. and Müller, P. (2001). Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA. ACM Press.

[Preparata and Shamos, 1985] Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA.

[Prusinkiewicz et al., 1988] Prusinkiewicz, P., Lindenmayer, A., and Hanan, J. (1988). Development models of herbaceous plants for computer imagery purposes. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 141–150, New York, NY, USA. ACM.

[Tarbell, 2008] Tarbell, J. (2008). Substrate algorithm. Gallery of Computation website. <http://complexification.net/gallery/machines/substrate/>, last visited Jul. 18, 2008.

[MAMMOTH Team, 2008] MAMMOTH Team (2008). Mammoth massively-multiplayer online game. Website. <http://mammoth.cs.mcgill.ca/>, last visited Jul. 18, 2008.

[Toussaint, 1983] Toussaint, G. T. (1983). Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON'83*, pages A10.02/1–4, Athens, Greece.

[Watson, 2006] Watson, B. (2006). Modeling land use with urban simulation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 185–251, New York, NY, USA. ACM Press.